APPLICATION FOR UNITED STATES PATENT

Inventor(s): John M. Haltmeyer
7535 Flamewood Drive
Clarksville, MD 21029
U.S. Citizen

Invention: THOROUGH OPERATION RESTRICTION

LAW OFFICES OF ROYAL W. CRAIG
10 N. Calvert St.
Suite 153
Baltimore, Maryland 21202
Telephone: (410) 385-2383

THOROUGH OPERATION RESTRICTION

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application derives priority from U.S. provisional application no. 60/268,522, filed February 14, 2001.

BACKGROUND OF THE INVENTION

1. Field of the invention

The present invention relates to the restriction of programs that a user can run on a computer and, more particularly, to a thorough method for restricting unauthorized operations by a user on a single workstation computer or by a user on a session in a multi-user environment, such as Microsoft Windows 2000 Terminal Services.

2. Description of the Background

A prominent problem with computers and networks today is lax security. Viruses are rampant. Typically, a user is emailed a script program that is really a virus (like the "Love Bug" virus) and unknowingly runs the program associated with the script. Obviously, this can cause a wide range of problems. Also, when a computer system is "locked down", hackers will often target the computer system to do nothing more than "break in". Unfortunately, existing methods tend to trade-off the level of security with demand on system resources. There is no high performance, completely secure method available today to resolve this problem. To understand the problem, a brief overview of the Windows architecture is helpful.

Figure 1 shows the Microsoft Windows 2000 architecture and illustrates the difference between user mode and kernal (or system) mode.

User mode is the least-privileged mode that Windows 2000 supports; it has no direct access to hardware and only restricted access to memory. For example, when software applications execute in user mode, they do so in a defined space with well-defined restrictions. They don't have direct access to hardware devices, and they can't touch parts of memory that are not specifically assigned to them.

On the other hand, system mode is a privileged mode. Those parts of Windows 2000 that execute in system mode, such as device drivers, have direct access to all hardware and memory. Other operating systems, including Windows NT, 3.1 and UNIX, also use analogous privileged and non-privileged modes. The services invoked in system mode are known as native advanced programming interface ("API"). The API is made up of about 250 functions that the operating system can access through software-exception system calls. A software-exception system call is a hardware-assisted way to change execution modes from user mode to system mode; it gives control over the data that passes between the two modes. Native API requests are executed by functions in system mode, known as system services.

As seen in Figure 1, the Executive components include the I/O Manager, Object Manager, Security Reference Monitor, Process Manager, Local Procedure Call Facility, and Virtual Memory Manager. Each Executive component has a specific operating system responsibility. Device drivers are dynamically added components that work closely with the I/O Manager to connect to specific hardware devices, such as disks and input devices. The Executive components use basic hardware functionality implemented in the System. Client-side

DLLs carry out tasks on behalf of their servers, but they execute as part of a client process. Given the foregoing overview, security will now be addressed.

In the Windows 2000 environment, there are two well known ways of controlling what programs (processes) specified users are allowed to run, and both have inherent flaws. One common way of marshaling processes is to use a technology known as a "filter driver". A filter driver is built to work in conjunction with the I/O Manager, intercepting all I/O within the disk system. The filter driver marshals those processes that are allowed to pass through and halts any processes that are unauthorized. Filter drivers run in system mode and they slow down the computer system significantly by monitoring all I/O with the particular subsystem. Moreover, filter drivers are processor intensive and are prone to single point of failure as well as problems with data loss.

Another means for marshaling applications (processes) is to "disallow" unauthorized programs. For example, US Patent No. 5,802,397 (IBM) shows a system for protection from unintended I/O access. An I/O protection array or list is used containing one-bit I/O keys. Each one-bit I/O key is used to disallow I/O accesses into an associated storage block. This method is not very practical. Only specified accesses or applications can be restricted. A "hacker" can write their own program to break a system, and it would not exist in the disallowed list of programs (processes), thus being allowed to be run by the hacker.

It would be greatly advantageous to overcome the problems associated with the two above-described methods with a user mode thorough operation restriction approach (ThOR), whereby a user can only run what an administrator has explicitly allowed the user to run, and all other processes will be terminated. By running in user mode, only the processes created by users would need to be validated as opposed to processor-intensive validation of all file I/O. This would decrease the processing requirements for marshaling user processes. Data loss due to program failure would be eliminated as system I/O is not interfered with, and any catastrophic failures would be constrained to a single user. This is an important feature when running a multi-user operating system like Windows 2000 Terminal Services.

SUMMARY OF THE INVENTION

In accordance with the above, it is an object of the present invention to provide a method for thoroughly restricting the applications (processes) that a user may run on a computer system.

The present invention is a Thorough Operation Restriction (ThOR) approach runs in user mode and not in system mode like filter drivers.

Generally, the method entails running two software modules in user mode, one of which maintains a list of allowed processes for each user and one of which monitors new processes as they are started. When a new process is started the monitoring module sends the process ID to the list module. The list module checks the ID against its list and kills the process if it is not authorized. This way, only processes created by users are validated as apposed to validating all file I/O (processor intensive). This decreases the processing requirements.

The software for Thorough Operation Restriction (ThOR) consists of two main compiled executables, or modules:

thor32.dll: This program is automatically attached to all new processes created.

Thor32.dll can be attached by the Windows 2000 program USER32, or alternatively, by the creation of something called a WndProc (the name for the message queue used to create a window).

mjolnir.exe: This is a security management executable that builds the allowed applications list, and terminates the unauthorized applications (processes).

ThOR can be implemented on any computer including multi-user systems (such as Windows 2000 Terminal Services) and generally comprises the following steps:

1st: The module thor32.dll is attached to all new processes by tying into the USER32 and by creating a WndProc hook. A hook is a function that monitors the 'goings on' inside the Windows operating system. The WndProc hook is called every time a certain event in windows occurs.[1]

2d: mjolnir.exe builds a list of allowed processes (allowed applications).

3d: When a new application is started, thor32.dll checks the process ID (this can be done by calling the API GetWindowThreadProcessId( ) function), and thor32.dll sends a message to mjolnir with the process ID of the process being examined.

4th: mjolnir.exe retrieves the name of the process as well as the path from the process ID received from thor32.dll. The name of the process is then checked against the list of allowed processes.

5th: If the process is allowed, mjolnir.exe answers the message back to thor32.dll, and thor32.dll marks the process as valid (no new messages are sent to mjolnir.exe from thor32.dll).

---

[1] See, Cummings, "An introduction to hook procedures", http://delphi.about.com/library/bluc/text/uc063001a.htm

6th: If the process is not authorized, mjolnir.exe kills the process.

7th: the above is repeated recursively, mjolnir continuing to receive messages from thor32.dll each time a new process is created.

8th: mjolnir.exe closes when it receives the WM_QUERYENDSESSION message, meaning the user is logging off.

By running in user mode, only the processes created by users are validated as apposed to validating all file I/O (processor intensive). This decreases the processing requirements for marshaling user processes. Operating system processes run in the security context of "SYSTEM" and are left alone. These processes, by definition, belong to the operating system and do not need to be monitored. Data loss due to program failure is also eliminated with ThOR, as system I/O is not interfered with. Since it runs in the user's context, ThOR eliminates the single point of failure common with filter driver technology. If ThOR has a failure, it is only for the single user. This is an important feature when running a multi-user operating system like Windows 2000 Terminal Services. ThOR builds a list of "authorized" applications (processes), and does not rely on the administrator to "disallow" a particular application (process). The user can only run what the administrator has explicitly allowed the user to run. All other processes will be killed.

BRIEF DESCRIPTION OF THE DRAWINGS

Other objects, features, and advantages of the present invention will become more apparent from the following detailed description of the preferred embodiment and certain modifications thereof when taken together with the accompanying drawings in which:

FIG. 1 is a system block diagram of the Microsoft Windows 2000 architecture.

FIG. 2 is a perspective block diagram of the general ThOR method of the present invention.

FIG. 3 is a flow chart of the Mjolnir.exe module.

FIG. 4 is a flow chart of the Thor32.dll module.

Appendix A is exemplary source code for thor32.dll.

Appendix B is exemplary source code for mjolnir.exe.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention is a software method for thoroughly restricting the applications (processes) that a user may run on a computer system. The method is implemented by software executables that run in user mode and not in system mode like filter drivers. The method of Thorough Operation Restriction (ThOR) consists of two main compiled executables, or modules:

1. thor32.dll: This executable program automatically attaches to all new processes created. Either by being attached by the Windows 2000 program USER32, or by the creation of something called a WndProc (the name for the message queue used to create a window). Appendix A is exemplary source code for thor32.dll.

2. mjolnir.exe: This executable program builds the allowed applications list, and kills the unauthorized applications (processes). Appendix B is exemplary source code for mjolnir.exe.

The combination can be implemented on any computer including multi-user systems (such as Windows 2000 Terminal Services) and generally comprises the following steps as shown in Fig. 2:

1st: at step 10, thor32.dll is attached to new processes by tying into the USER32 and by creating a WndProc hook (See Appendix A, page 2, line 10)

2d: at step 20, mjolnir.exe builds a list of allowed processes (allowed applications). (See Appendix B, page 6, lines 15 et seq.)

3d: at step 30, when a new application is started, thor32.dll sends a message to mjolnir with the process id of the process being examined. (See Appendix A, page 1, lines 40 et seq.)

4th: at step 40, mjolnir.exe retrieves the name of the process as well as the path from the process id received from thor32.dll. (See Appendix B, page 9, line 35).

5th: at step 50, if the process is allowed, the message is answered and thor32.dll marks the process as valid (no new messages are sent to mjolnir.exe from thor32.dll). (See Appendix A, page 2, lines 29-31).

6th: at step 60, if the process is not authorized, mjolnir.exe kills the process. (See Appendix B, page 12, line 9).

7th: at step 70, mjolnir continues receiving messages from thor32.dll each time a new process is created.

8th: at step 80, mjolnir.exe closes when the user is logs off.

FIG. 3 is a more detailed flow diagram of <u>mjolnir.exe</u>, the executable program that builds the allowed applications list, and kills unauthorized applications (processes).

At step 100 a user logs on to the computer.

At step 110 the executable mjolnir.exe is automatically started by the Windows userinit process.

At step 120, after startup, mjolnir.exe waits for a pre-defined time to allow logon scripts to complete running.

At step 130, after the delay has expired, mjolnir.exe builds an "allowed applications" list in memory. This list includes the full path to the executable as well as the number of times a user is allowed to run it. The list data can reside in a database and can be based on group membership, or it can be read from a configuration file.

At step 140 mjolnir.exe now installs something called a hook to attach thor32.dll to all new processes that create a graphical interface. A hook is a mechanism by which a function can intercept events (messages, mouse actions, keystrokes) before they reach an application. In order for Windows 2000 to call a function like thor32.dll, the latter must be attached to a Windows hook. Attaching the function to a hook is known as setting the hook. When a hook has a function attached and an event occurs that triggers the hook, Windows calls the function. This action is known as calling the hook. To maintain and access the thor32.dll function, the present method uses the SetWindowsHookEx and the UnhookWindowsHookEx functions. The SetWindowsHookEx function adds the function to the hook so that the function can act on events and, in some cases, modify or discard them. In the present case, Thor32.dll is a filter function that receives events. In accordance with the present invention, the SetWindowsHookEx is used to attach thor32.dll to all new processes that create a graphical interface.

After setting the hook, at step 150 mjolnir.exe now waits for a message from the thor32.dll program that is attached to all new processes.

FIG. 4 is a more detailed flow diagram of <u>thor32.dll</u>.

At step 300 a new process is created and, as previously explained, thor32.dll is automatically attached to the process. At step 310 the program checks to see if the process is linked to User32.dll. If so, thor32.dll is attached to the process at step 310 using USER32.DLL.

If not, thor32.dll is attached to the process at step 340 via step 330, wherein a hook is loaded. The hook is loaded by exporting a function (_LoadThorA) from thor32.dll, and the LoadThorA function in turn calls the SetWindowsHookEx function. The SetWindowsHookEx function takes three arguments:

1. An integer code describing the hook to which to attach thor32.dll

2. The address of thor32.dll.

3. The InstanceHandle for which the hook is to be installed. The InstanceHandle is used, not the thread ID, because the hook is called from the exported function (_LoadThorA) in thor32.dll, instead of the mjolnir.exe (where the threadID would otherwise be needed). This simplifies the process of loading the hook, since thor32.dll already knows what thread it is running in. The installed filter function will be called only in the context of the specified thread. The specific call to the SetWindowsHookEx function appears as follows:

SetWindowsHookEx(WH_CALLWNDPROC, (HOOKPROC)GetMsgProc, InstanceHandle, 0).

Referring now to step 350, after thor32.dll is attached to the new process, it sends a message to mjolnir.exe by calling the SendMessage( ) function. One of the parameters of the message being sent is the process id (the unique identifier of the process) of the current process.

Referring back to FIG. 3, we left at step 150 with mjolnir.exe waiting for the above-described message.

At step 160, the message is received from thor32.dll.

At step 170, every time a message is received from thor32.dll to validate a process id, mjolnir.exe checks it's list of authorized applications and verifies that the process is allowed to run.

If the process is allowed to run, at step 180 the message is answered (also see step 370 of Fig. 4).

If the process is not authorized, at step 190 the process is killed (also see step 380 of Fig. 4).

Mjolnir continues this loop until it receives the WM_QUERYENDSESSION message meaning a logoff has ocured (step 200). Once the logoff message has been received, mjolnir.exe closes (step 210).

In the foregoing manner, the present ThOR method builds a list of "authorized" applications (processes), and does not rely on the administrator to "disallow" a particular application (process). The user can only run what the administrator has explicitly allowed the user to run. All other processes will be killed. Moreover, since both thor32.dll and mjolnir.exe run in user mode, only the processes created by users are validated as apposed to validating all file I/O (processor intensive). This decreases the processing requirements for marshaling user processes. Operating system processes run in the security context of "SYSTEM" and are left alone. These processes, by definition, belong to the operating system and do not need to be monitored. Data loss due to program failure is also eliminated with ThOR, as system I/O is not interfered with. Since it runs in the user's context, ThOR eliminates the single point of failure common with filter driver technology. If ThOR has a failure, it is only for the single user. This is an important feature when running a multi-user operating system like Windows 2000 Terminal Services.

Having now fully set forth the preferred embodiments and certain modifications of the concept underlying the present invention, various other embodiments as well as certain variations and modifications of the embodiments herein shown and described will obviously occur to those skilled in the art upon becoming familiar with said underlying concept. It is to be understood, therefore, that the invention may be practiced otherwise than as specifically set forth in the appended claims.